STRUCTURI DE CONTROL

1. Structuri de control la nivel de instrucțiune

- Precizează ordinea în care se efectuează instrucțiunile individuale ale programului.
- Se grupează în trei categorii:secvența, selecția şi repetiția.

Secvența.

- Este cea mai simplă structură de control.
- În Pascal: *begin* … *end*;
  În C: { … }

Selecția.

- Permite alegerea unei alternative între două sau mai multe posibile.
- Instrucțiunea **if** – în Pascal, Algol

```
if condiție then
        Secvență_de_instrucțiuni
else
        Secvență_de_instrucțiuni
```

- Instrucțiunea **case** – în Pascal, Ada, Algol68
- Instrucțiunea **switch** – în C

- Exemplu în Pascal standard:

```
if nota <= 10 then
    case nota of
        1,2,3,4 : writeln('picat');
          5,6,7 : writeln('promovat');
            8,9 : writeln('bine');
             10 : writeln('excelent');
    end
else
    writeln('nota gresita');
```

- Variantă în Ada:

```
case nota of
        when 1..4 ⇒ put_line("picat");
        when 5 |6 |7 ⇒ put_line("promovat");
        when 8 |9 ⇒ put_line("bine");
        when 10 ⇒ put_line("excelent");
        when others ⇒ put_line("nota gresita");
end case ;
```

Repetiția.

- Reprezintă mecanismul de bază pentru efectuarea unor calcule complexe în limbajele de programare imperative.
- Structuri controlate prin condiție:

- În Pascal:
  while *conditie* ***do***
  instructiune

  repeat
  secventa de instructiuni
  until *conditie*

- Structuri controlate prin contor:

- În Fortran- instrctiunea DO
- Exemplu general:
  for *variabila:=val_initiala* ***to*** *val_finala* ***step*** *val_pas* ***do***
  instructiune
- În Pascal (ca şi în Ada) s-a renunțat la clauza **step**.

- În Algol68 şi PL/I:
  for *variabila* ***from*** *val_i* ***by*** *val_p* ***to*** *val_f* ***while*** *conditie* ***do***
  secventa de instructiuni
  od

- Părăsirea forțată a ciclului:
- În Ada: **exit**
  exit *nume*
- În C: **break**
  continue

2. Structuri de control la nivel de subunitate

- **Subprograme.**

- efecte laterale – modificare provocată de un subprogram asupra unei entități care nu este locală subprogramului. Sunt deranjante, mai ales în cazul funcțiilor:
  v := a + f(a,b) + c;

- pseudonime – un obiect poate fi referit, la un moment dat, prin două sau mai multe nume diferite.

Pot să apară:

✓ în conexiune cu procedurile, în cazul transmisiei prin adresă a parametrilor:

```
var y:integer;
---------------
procedure p(x);
begin
   x:=2*x;
   y:=x+y;
end;
---------------
y:=1;
p(y);
---------------
```

✓ atunci când două sau mai multe argumente (transmise prin adresă) reprezintă acelaşi obiect.

```
var  z:integer;
-----------------
procedure p(x,y:integer);
begin
    x:=2*x;
    y:=2*y;
end;
-----------------
z:=3;
p(z,z);
-----------------
```

✓ atunci când se transmit ca parametri o structură sau componente ale structurii respective.

```
type t = array [1..100] of real;
var tab:t;
procedure  pp(x:integer; y:t);
begin
-----------------
end;
------------------
pp(tab[3], tab);
------------------
```

✓ atunci când două argumente sunt două elemente ale aceluiaşi tablou.

```
p(t[i], t[j]);
```

pseudonime apar doar dacă i=j

Tratarea excepțiilor în Ada.

- Există 5 excepții predefinite:
  - **constraint_error:** violarea limitelor unui subdomeniu, referirea unui câmp ilegal într-un articol cu variante, referirea unui pointer cu valoarea **null**;
  - **numeric_error**: depăşirea aritmetică;
  - **storage_error**: depăşirea spațiului de memorie;
  - **select_error, tasking_error**: aspecte de concurență.

- Declararea excepțiilor:
  eroare, sfarsit: ***exception****;*

- Declanşarea unei excepții:
  raise *eroare;*

- Exemplu de tratare a excepțiilor:

```
function f(x:float) return float is
   negativ:exception;
begin
   if x<0 then
         raise negativ;
   else
         return 1/sqrt(x);
   end if;
exception
   when numeric_error =>
               return 0;   -- returnează 0 dacă x este 0
   when negativ =>
               return –1; -- returnează -1 dacă x e negativ
end  f;
```

```
package stack is
        eroare: exception;
        type stiva(nr_max:integer) is limited private;
        function pop(s:in out stiva) return integer;
        procedure push(s:in out stiva; x: integer);
        function top(s:stiva) return integer;
        procedure init(s:out stiva);
private
        type stiva(nr_max:integer) is
                record
                   tab_st:array(1..nr_max) of integer;
                   ind:integer;
                end record;
end stack;
package body stack is
        function empty(s:stiva) return boolean is
        begin  ----------  end  empty;
        function overflow(s:stiva) return boolean is
        begin  ----------  end  overflow;
        function pop(s: in out stiva) return integer is
        begin
                if not empty(s) then
                        s.ind:=s.ind-1;
                        return s.tab_st(s.ind);
                else    raise eroare;
                end if;
        end  pop;
        procedure push(s: in out stiva; x:integer) is
        begin
                if not overflow(s) then
                        s.tab_st(s.ind):=x;
                        s.ind:=s.ind+1;
                else    raise eroare;
                end if;
        end  push;
```

```
        function top(s:stiva) return tip_el is
        begin
                if not empty(s) then
                        return s.tab_st(s.ind-1);
                else
                        raise eroare;
                end if;
        end  top;
        function init(s: out stiva)  is
        begin
                ---------------
        end  init;
begin
end stack;
```

- Propagarea excepției în procedura apelantă:

```
procedure st is
        use stack;
        stv:stiva(100);
        ----------------------
        init(stv);
        ----------------------
        push(stv,10);
        ----------------------
        i:=top(stv);
        ----------------------
exception
        when eroare =>
                put_line("eroare in stiva");
                while not empty(stv) loop
                                put(pop(stv));
                end loop;
   end st;
```

Tratarea excepțiilor în C#

- Mecanismul de tratare a excepțiilor din C# este similar cu cel din C++ şi Java.
- Excepțiile sunt reprezentate prin *clase.*

Clasa `System.Exception`

- Toate clasele care reprezintă excepții trebuie să derive din clasa predefinită `Exception`, care face parte din spațiul de nume `System`.
- Clasele care derivă din `Exception` sunt:
    - `SystemException`
    - `ApplicationException`
- Acestea implementează cele două categorii generale de excepții definite de limbajul C#:
    - cele generate de motorul de execuție
    - cele generate de programele de aplicație.

➢ Limbajul C# defineşte câteva tipuri de *excepții standard* care sunt derivate din `SystemException`. Sunt generate de motorul de programare atunci când se produc erori la execuția programului. Cele mai frecvent utilizate sunt:

Excepția	Semnificația
`ArrayTypeMismatchException`	Tipul valorii memorate este incompatibil cu tipul tabloului
`DivideByZeroException`	Încercare de împărțire la zero
`IndexOutOfRangeException`	Indexul din tablou depăseşte marginile
`InvalidCastException`	Cast incorect la execuție
`OutOfMemoryException`	Apelul lui `new` eşuează din cauza memoriei insuficiente
`OverflowException`	Depăşire aritmetică
`StackOverflowException`	Depăşirea capacității stivei

➢ *Programatorul poate defini* propriile clase care reprezintă excepții, derivându-le din `ApplicationException`.

Noțiuni de bază despre tratarea excepțiilor

Cuvintele cheie pentru tratarea excepțiilor în C# sunt:

try
catch
throw
finally

Cum se tratează excepțiile ?

- Corpul blocurilor **try** (încercare) cuprinde instrucțiunile care trebuie verificate pentru apariția excepțiilor.
- Dacă pe parcursul execuției unui bloc **try** se produce o excepție, aceasta este lansată (**throw**).
- Utilizând **catch**, programul poate intercepta această excepție, pe care apoi o tratează în conformitate cu cerințele aplicației.

- Excepțiile sunt lansate:
  - în mod automat de către motorul de execuție din C#;
  - manual, utilizând cuvântul cheie **throw**.

- Codul care trebuie neapărat executat la ieşirea dintr-un bloc **try** trebuie inclus într-un bloc **finally** (de final).

- Pentru a intercepta toate excepțiile, indiferent de tipul lor, folosim instrucțiunea **catch** fără nici un parametru. Aceasta creează o rutină „universală” care interceptează şi tratează toate excepțiile.

Exemplu simplu de tratare a unei excepții.

```
using System;

class Exemplu1 {
  public static void Main() {
   int [] nums = new int[4];

   try {
     Console.WriteLine(“Inainte de generarea
         exceptiei.”);

   // Generam o exceptie de depasire a valorii
   // indicelui.
     nums[7] = 10;
     Console.WriteLine(“Mesaj care nu se
                        afiseaza”);
   }
   catch(IndexOutOfRangeException) {
   // Interceptam exceptia
     Console.WriteLine(“Indexul depaseste
                        marginile!”);
   }
   Console.WriteLine(“Dupa instructiunea catch.”);
  }
}
```

Textul afişat de program este:

```
Inainte de generarea exceptiei.
Indexul depaseste marginile!
Dupa instructiunea catch.
```

Exemplu care utilizează instrucțiunea `throw` pentru a lansa manual o excepție.

```
using System;

class Exemplu2 {
  public static void Main() {
    try {
      Console.WriteLine(“Inainte de throw.”);
      // lansarea exceptiei
      throw new DivideByZeroException();
    }
    catch(DivideByZeroException) {
      // interceptam exceptia
      Console.WriteLine(“Exceptie interceptata.”);
    }
```

Rezultatul programului este:

```
Inainte de throw.
Exceptie interceptata.
Dupa blocul try/catch.
```